ECHO Services Refactor

> ECHO has been replaced by the **Common Metadata Repository (CMR)**, a high-performance, high-quality, continuously evolving metadata system that catalogs all data and service metadata records for the EOSDIS system and will be the authoritative management system for all EOSDIS metadata.
>
> The information contained within this ECHO wiki is now archived for historical reference. Please navigate to the **CMR wiki pages**, or to the **CMR Overview page** on **Earthdata**.

Design

Abstract:

In order to provide a more streamlined, maintainable, and extensible system for data acquisition (order, service, direct download) requests invoked via Reverb, and to enable the sharing of code between Reverb and other clients (e.g. EDSC), a new top level project will be developed. This project will provide a unified API for creating, editing, submitting, and monitoring data acquisitions. The design will take into consideration both the Reverb New Order Workflow(NOW) and the EDSC workflow and must integrate with both workflows. Since multiple clients will be sharing this API, we will not make any assumptions about the View or user interface (except the assumption that all clients will use the echo_forms_plugin for option selection).

Rails Endpoints:

NOTE: the endpoints in blue below will likely be implemented in the client (Reverb or EDSC).

URL template	Method	Purpose
/service_requests/new	POST	Returns an HTML page where the user can set options for a new request containing the catalog items passed in the POST body. This would primarily be called from the Reverb shopping cart. In this use case, the shopping cart contents, as well as user and session information would be rolled up into the POST body. The POST should also contain the Reverb search criteria, for possible use in pre populating values in the service options.
/service_requests/:item_id/new	GET	Build a new single item request. This would be a new use case, but it may be useful to allow a user to submit a request without going through the shopping cart.
/service_requests/submit	POST	Build and submit a service request. Called from the /service_requests/new page. Returns service status URL as the location header. Request body contains items and options for request.
/service_requests/:request_id	GET	Get the status of a request.
	PUT	Update the status of a request.
/service_options/:item_id(.:format)	GET	xml: Get the option definition(s) (echo form) xml for the dataset or granule. html: Render the option definition(s) for the dataset or granule. If there are multiple relevant option definitions, they will all be displayed. A query param '?num=x' can be used to specify only one

Models:

- **DataRequest** - Base model type shared by Orders and Service Requests. Defines basic functionality and sets up the interface which all orders and service requests will conform to.
  - **Download** -
  - **Order** - Placeholder for potentially integrating Orders and Service Requests in the future.
  - **ServiceRequest** - Common service functionality
    - **SSWServiceRequest** - There are currently no SSW services, only ESI, which is an extension of SSW, but this model will separate the APIs and allow us to more easily integrate SSW services. Functionality which would be shared between the APIs is defined her.
      - **ESIServiceRequest** - ESI specific functionality (that is not shared by SSW)

Data location/access:

Current and proposed location of data used by the API and how it is/will be accessed

- Service Option Definition/Assignment - DEV_52_BUSINESS.SERVICE_OPTION_DEFINITION/ASSIGNMENT- Accessed via echo-rest
- Order Option Definition - DEV_52_BUSINESS.EJB_OPTION_DEF - accessed via echo-rest
- Order Option Assignment - ?
- Servicability - DEV_52_BUSINESS.SERVICE_OPTION_ASSIGNMENT
- Orderability -
- Service - DEV_52_REVERB.SERVICE_ORDERS - NEW SYSTEM: Tables in TBD new schema
- Order - DEV_52_BUSINESS.EJB_ORDER (& EJB_PROVIDER_ORDER?)
- Downloads - URLs in metadata. possibly tracked in new schema.

Use Cases:

Perform service on contents of shopping cart (existing nominal use case):

1. User performs search and adds catalog items (datasets and granules) to the shopping cart.
2. User navigates to the shopping cart and presses the "Perform Service" button.
3. Reverb packs up the shopping cart contents, as well as the user information, token ID, spatial/temporal query params, etc. and performs a POST to /service_requests/new.
4. The user is presented with a page which mirrors the current New Order Workflow page,
5. The user adds any necessary User Information (name and email address) or logs in if they are not already.
6. The user may remove items from their order.
7. The user sets the service processing options, which may have been pre populated with information already (e.g. email address, spatial/temporal subset bounds pulled from search criteria).
8. Once all required information is set, the "Submit" button becomes active.
9. The user clicks the "Submit" button which performs a POST to /service_requests/submit. The POST body contains all of the selected catalog items, along with their option selections and user information.
10. The service_request controller builds one or more ServiceRequest objects (items will be split up based on shared service option selections).
11. The controller will then submit each ServiceRequest
    a. For SSWServiceRequests(Including ESIServiceRequests), the submittal will entail:
        i. Pull parameters from the service options selections
        ii. Marshall the selected parameters into an SSW/ESI formatted request using an ERB template.
        iii. Perform a POST to the configured service endpoint.
12. The URL /service_requests/:request_id will be returned to Reverb. Reverb will then display this page and refresh it periodically to get updated status.
13. The user can also navigate to the reverb service status page to be directed to service status URL (/service_requests/:request_id) for any existing request.

Submit request for a single catalog item (potential new functionality):

1. User is on a dataset or granule search result page in Reverb.
2. User clicks on the 'Services Available' gear icon next to the catalog item.
3. User is presented with some information about available services as well as a 'Perform Service' button.
4. User presses the 'Perform Service' button.
5. Reverb packs up the selected item id, as well as the user information, token, spatial/temporal query params, etc. and performs a POST /service_requests/:item_id/new.
6. Pick up the nominal use case above at step 4.

View options definition for item (potential new functionality):

1. DAAC Operator uploads a service option definition and assigns it to a dataset (e.g. via SDPS EGI GUI, PUMP, or ECHO web services).
2. Operator navigates to /service_options/:item_id.xml and sees the ECHO forms XML for all option definitions for the specified item.
    a. Operator can alternatively navigate to /service_options/:item_id.xml?num=x to view only one option definition if multiple exist.
3. Operator navigates to /service_options/:item_id.html and is presented with an HTML page containing fully functional forms for the specified item. This page would look similar to the existing reverb new_test_form page and would display the dynamic model output for each form, facilitating debugging of forms.
    a. Operator can alternatively navigate to /service_options/:item_id.html?num=x to view only one option definition if multiple exist.

Issues:

- Need to make sure we can share session information across the reverb and service endpoints. Hopefully we can accomplish this by passing a token in the initial request.
- If we replicate the New Order Workflow in the service project, but do not move Order processing into the same project yet, we will end up with duplicate code. If we cant find a way to share this code. We at least need to put some thought to making sure it gets cleaned up if/when orders move to this new project as well.

Appendix

Existing Reverb Service Endpoints

Relevant endpoints from the existing implementation are shown below for reference.

Service Entries (i.e. service implementations)

Name	Method	URL template	Controller method	Purpose	Keep?
service_entries	GET	/service_entries(.:format)	service_entries#index	List service entries	

Service Orders

Name	Method	URL template	Controller method
submit_service_order	POST	/service_orders/:id/submit(.:format)	service_orders#submit
review_service_order	GET	/service_orders/:id/review(.:format)	service_orders#review
receipt_service_order	GET	/service_orders/:id/receipt(.:format)	service_orders#receipt
service_order_validate_option_selections	GET	/service_orders/:service_order_id/validate_option_selections(.:format)	service_orders#validate_option
search_service_orders	GET	/service_orders/search(.:format)	service_orders#search
metrics_service_orders	GET	/service_orders/metrics(.:format)	service_orders#metrics

service_orders	GET	/service_orders(.:format)	service_orders#index
	POST	/service_orders(.:format)	service_orders#create
edit_service_order	GET	/service_orders/:id/edit(.:format)	service_orders#edit
service_order	GET	/service_orders/:id(.:format)	service_orders#show
	DELETE	/service_orders/:id(.:format)	service_orders#destroy
service_order_service_order_item	PUT	/service_orders/:service_order_id/service_order_items/:id(.:format)	service_order_items#update

Test Endpoints

Name	Method	URL template	Controller method	Purpose	Keep?
esi_endpoint_index	GET	/esi_endpoint(.:format)	esi_endpoint#index	Display test page (broken)	
	POST	/esi_endpoint(.:format)	esi_endpoint#create	Perform test service request	
esi_endpoint	GET	/esi_endpoint/:id(.:format)	esi_endpoint#show	Return status of test service request	
	DELETE	/esi_endpoint/:id(.:format)	esi_endpoint#destroy	Cancel test service request	
esi_requests	GET	/esi_requests(.:format)	esi_requests#index	Placeholder (401)	
esi_request	GET	/esi_requests/:id(.:format)	esi_requests#show	?	
ssw_endpoint_index	GET	/ssw_endpoint(.:format)	ssw_endpoint#index	Display test page(broken)	
	POST	/ssw_endpoint(.:format)	ssw_endpoint#create	Perform test service request	
ssw_endpoint	GET	/ssw_endpoint/:id(.:format)	ssw_endpoint#show	Return status of test service request	
	DELETE	/ssw_endpoint/:id(.:format)	ssw_endpoint#destroy	Cancel test service request	

Orders

Name	Method	URL template	Controller method	Purpose	Workt

orders	GET	/orders(.:format)	orders#index	List orders for current user	
	POST	/orders(.:format)	orders#create	Creates an order form the shopping cart	
search_orders	GET	/orders/search(.:format)	orders#search	Find order by ID	
new_order	GET	/orders/new(.:format)	orders#new	Set order Options (NOW)	NOW
edit_order	GET	/orders/:id/edit(.:format)	orders#edit	legacy?	
order	GET	/orders/:id(.:format)	orders#show	Get order information	
	DELETE	/orders/:id(.:format)	orders#destroy	Cancel order	
show_granules_order	GET	/orders/:id/show_granules(.:format)	orders#show_granules	List granules in specified dataset which are in the cart. ':id' is thrown away. Should we move this under cart?	
?	POST	/orders/:id/submit(.:format)	orders#submit	Submit the order. In NOW, this includes creating the order.	both
submit_order	GET	/orders/:id/submit(.:format)	orders#view_order_submission	View order receipt	NOW
order_validate_option_selections	GET	/orders/:order_id/validate_option_selections(.:format)	orders#validate_option_selections	Verify all options are set	
remove_selected_order	POST	/orders/:id/remove_selected(.:format)	orders#remove_selected	remove selected order items	
remove_all_order	POST	/orders/:id/remove_all(.:format)	orders#remove_all	remove all order items	
order_order_items	POST	/orders/:order_id/order_items(.:format)	order_items#create	Add order item	legacy
	DELETE	/orders/:order_id/order_items(.:format)	order_items#destroy	Remove order item	legacy
order_order_item	PUT	/orders/:order_id/order_items/:id(.:format)	order_items#update	Update order item	legacy